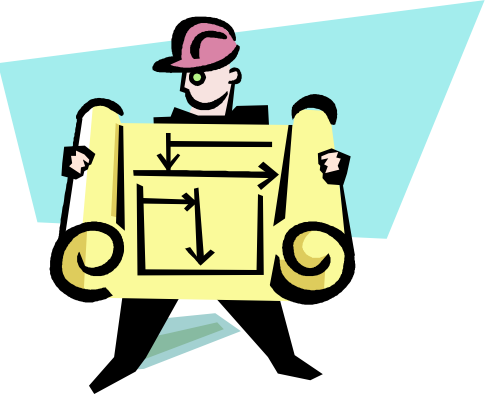
Dokumentationsverfahren für Software Architekturen

Jan Weinschenker
jan.weinschenker@informatik.haw-hamburg.de

Agenda

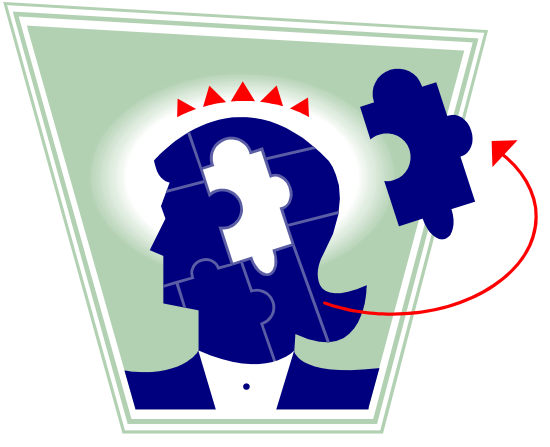
- Motivation
- Grundlagen
- „Views & Beyond“
- Zusammenfassung



Motivation

Wozu der Aufwand?

- **Kommunikationsgrundlage**

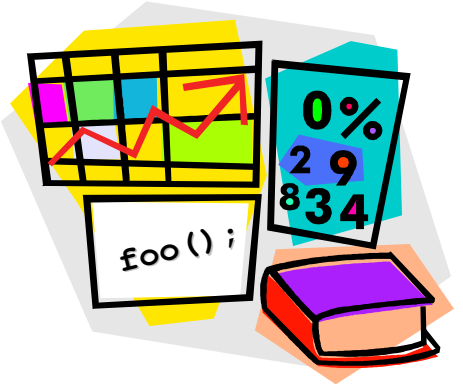


Wozu der Aufwand?

- Damit man weiß, was vor sich geht
  - Planvolles Vorgehen
  - Sinn und Zweck der Software?
  - Laufzeitverhalten?
  - Ressourcenverbrauch?

Wozu der Aufwand?

- (Formale) Architektur-Analyse
  - Beschreibungssprachen / ADL
  - ATAM[1]-Methode [CKK02]



[1] *Architecture Tradeoff Analysis Method*

Motivation

- Je komplexer ein Projekt, desto …
- mehr Beteiligte, desto …
- mehr Kommunikation ist erforderlich

- Verteilte Systeme sind i.d.R. sehr komplex

Grundlagen

Was ist Software Architektur?

Die Software Architektur eines Programms oder Informationssystems ist die Struktur oder sind die Strukturen des Systems, welche

- Software Elemente,
- die extern sichtbaren Bestandteile dieser Elemente,
- die Beziehungen unter ihnen

beinhalten. [BCK03]

Was ist Software Architektur?

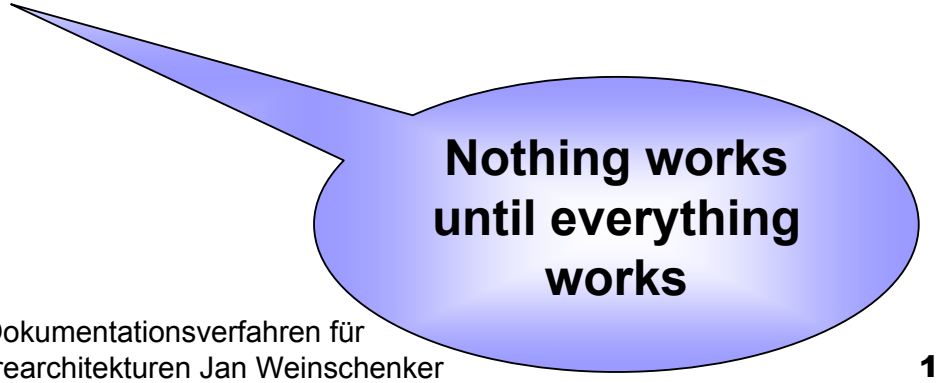
- *The fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution* [IEEE1471]

- 40+ weitere Definitionen: [SEI05]

Was ist Software Architektur?

- Eine Grundlage für Arbeitsteilung
- Salopp gesagt:
  - „Divide and Conquer“
  - „Now mind your own Business“
  - „So how do these things work together?“

[CBB+03]



Stakeholder

Wer ist am Projekt beteiligt?

- Projektteam(s)
  - Entwickler / Tester
  - Architekt(en)
  - Vorwissen / Interessen sind unterschiedlich
- Kunde(n)
  - Geldgeber
  - Anwender
- Chef
- Externe
- Nicht-Techniker
  - Marketing
  - Verkauf
  - Controlling
  - Anwender
- …

Stakeholder

Welche Information ist für wen interessant?

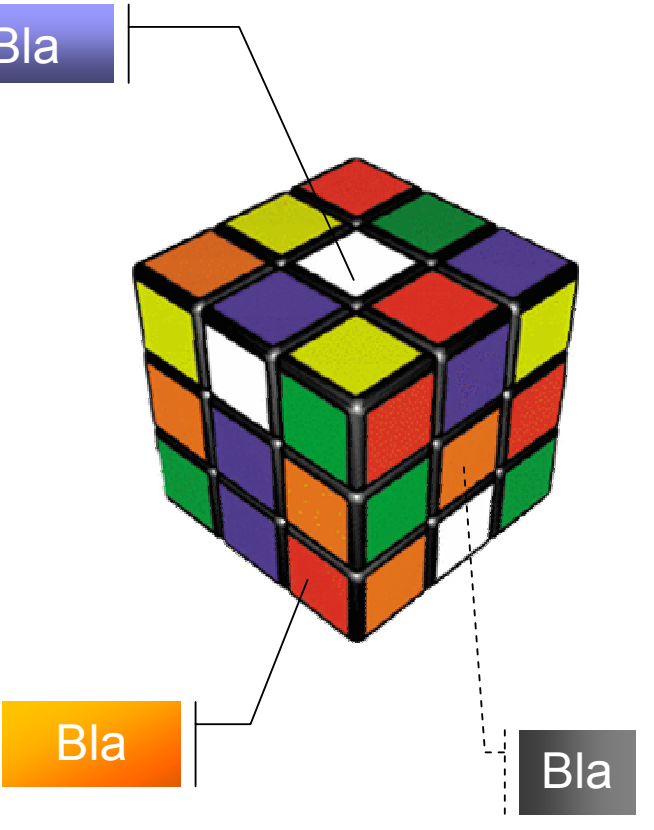
- Notation: Formal oder nicht Formal?
- Welche Details …
  - … sind von Bedeutung?
  - … sind überflüssig?
  - … bergen Risiken?
- Wer braucht eher den groben Überblick?
  - In wie weit abstrahieren?

Dokumentation & Beschreibung

- Dokumentation
  - Soll der zwischenmenschlichen Kommunikation dienen
- Beschreibungssprachen sind
  - Im Sinne dieses Vortrags formale Notationen
  - Bestandteil der Dokumentation
  - Von Fachleuten für Fachleute

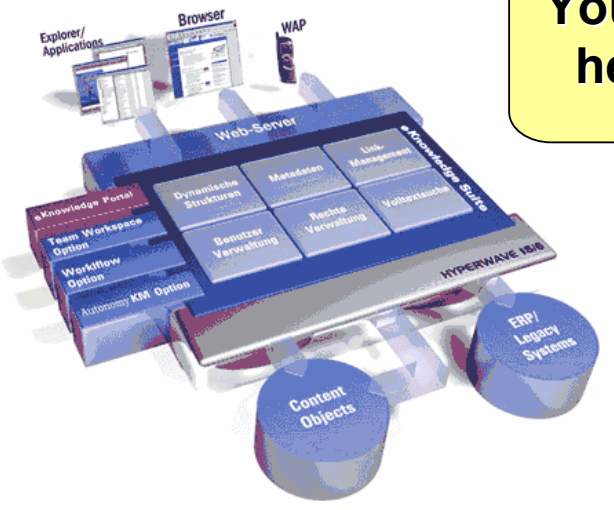
Sichten / Views

- „Documenting an architecture is a matter of documenting the relevant views and then adding documentation that applies to more than one view.“ [Cle05]

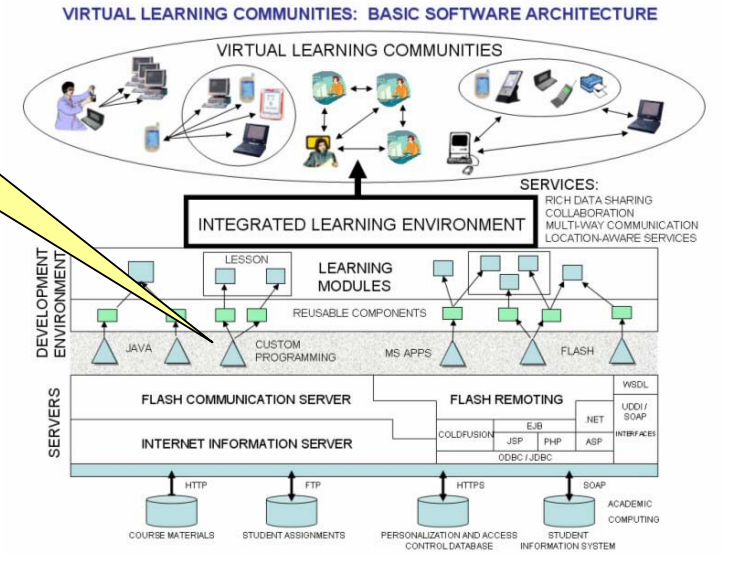


Dokumentation & Coding

- *Architecture is what makes the sets of parts work together as a successful whole. Architecture documentation is what tells developers how to make it so.* [CBB+03]

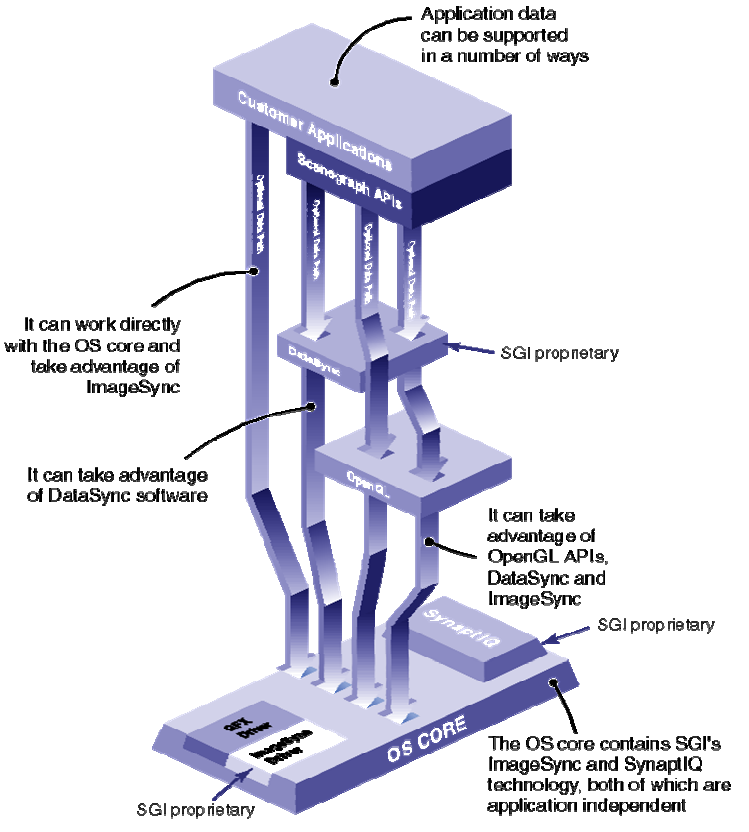


You are here!



Dokumentation & Information

- In vielen Fällen soll Dokumentation nur einen schnellen Überblick verschaffen
- Ohne:
  - Formale Notationen
  - Vollständigkeit



ADL - Beschreibungssprachen

- Aesop
- Adage
- Meta-H
- C2
- Rapide
- SADL
- UniCon
- Wright

- ACME
  - xACME
  - ADML
- UML 1.* ?
- UML 2.0 ?
- Informal

Hauptteil

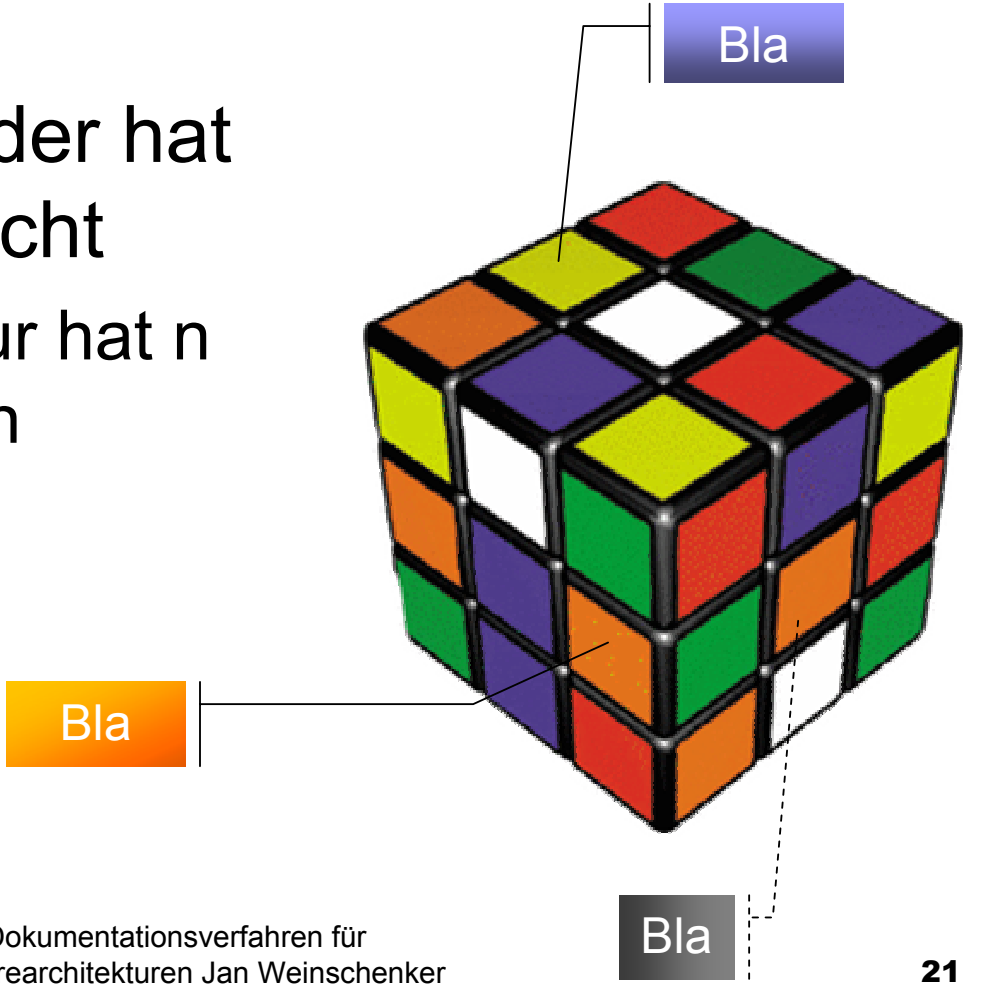
Dokumentationsverfahren

- Views & Beyond (V&B)
  - Umfassender Ansatz zur Dokumentation von softwareorientierten Architekturen
  - Carnegie Mellon University [CBB+03, SEI05]

- ANSI-IEEE 1471-2000
  - „Best practice“, nicht auf Software beschränkt
  - [IEEE1471]

Architektonische Sichten

- Jeder Stakeholder hat seine eigene Sicht
  - Jede Architektur hat n Seiten / Sichten



Sichten in V&B: Viewtypes

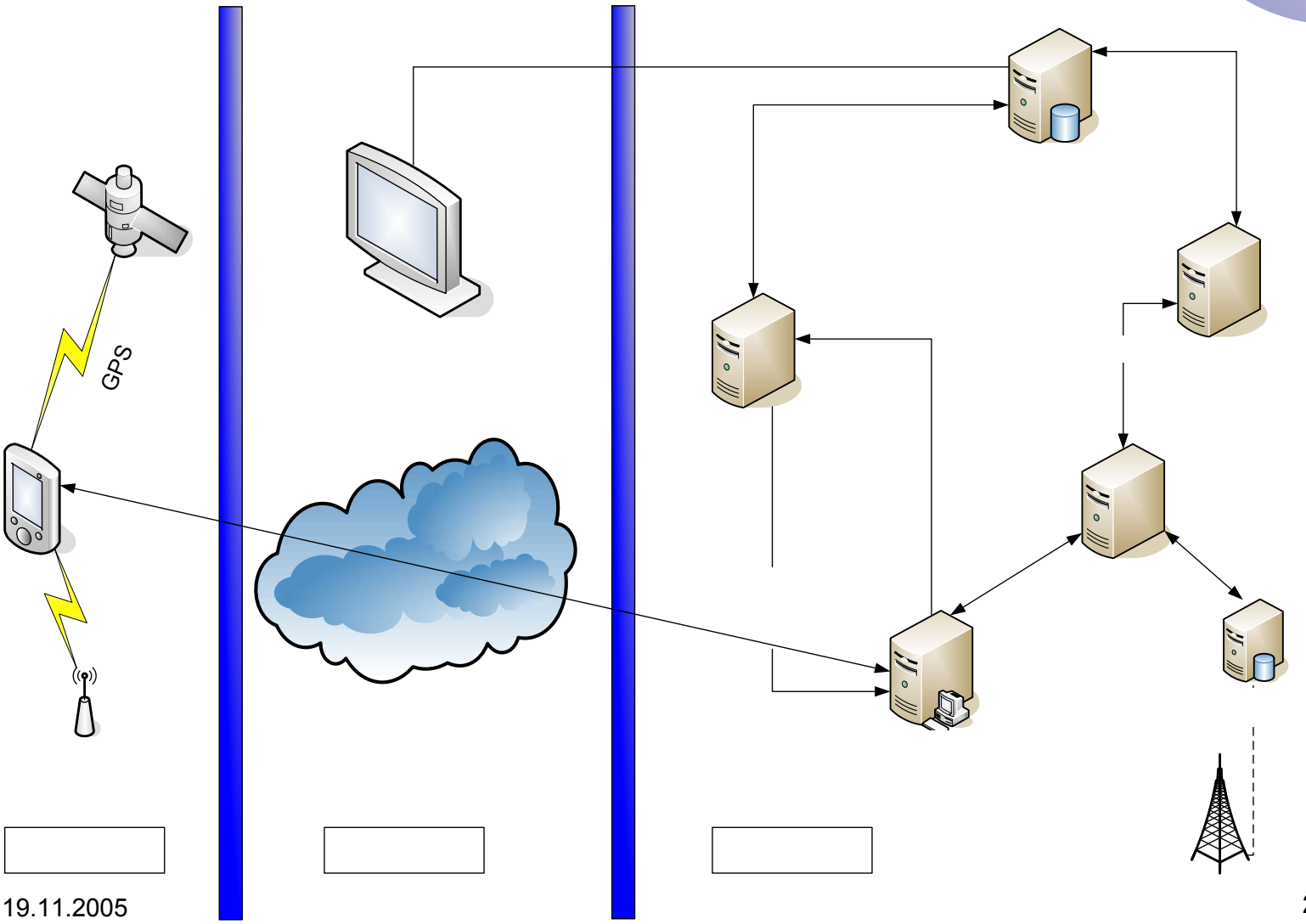
- Module Viewtype
  - Statische Sicht der Elemente
  - Implementationseinheiten
- Component-And-Connector Viewtype
  - Laufzeitverhalten der Elemente
  - Interaktion der Elemente
- Allocation Viewtype
  - Interaktion mit der (Nichtsoftware-)Umgebung
  - Menschen, Hardware, Dateisysteme, …

Module Viewtype

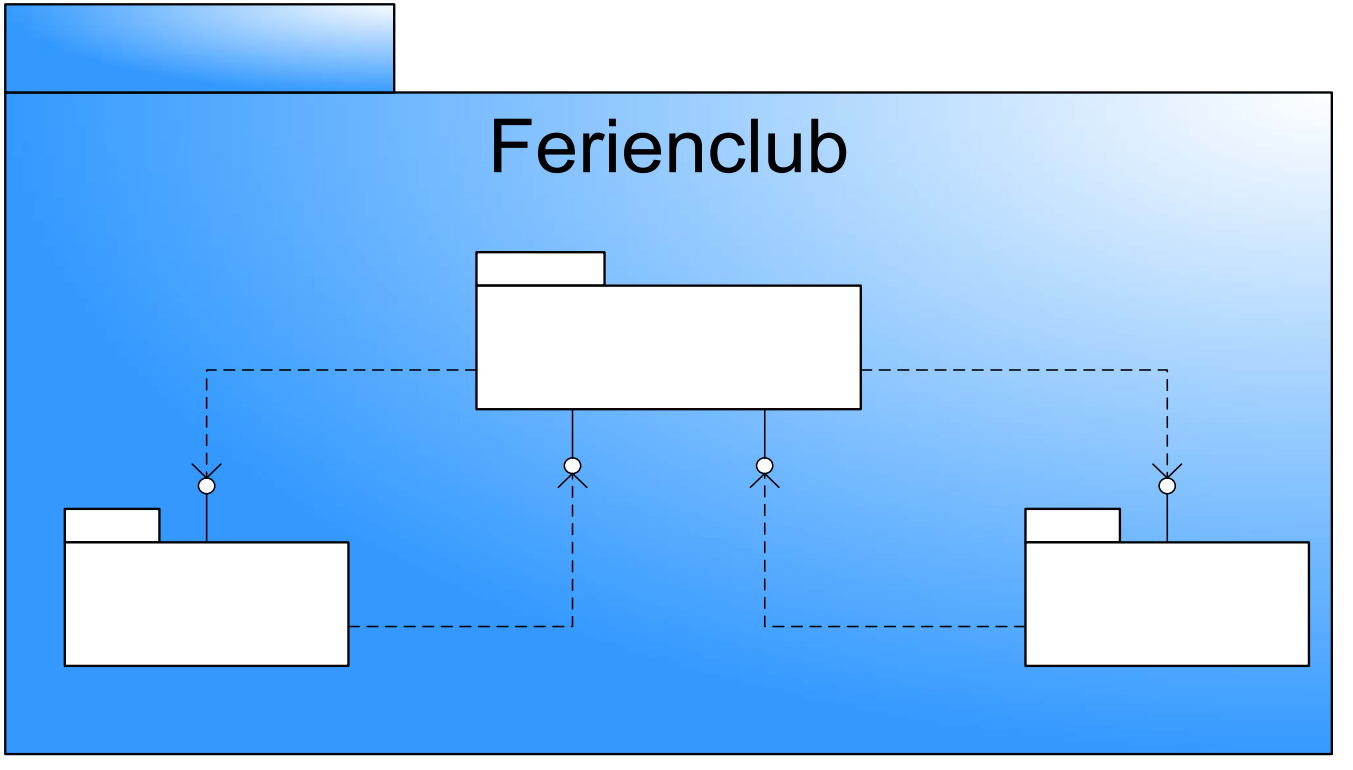
- Konzeptioneller Überblick
- Zerlegung der Architektur in implementierbare Einheiten
- Statische Strukturen

- Elemente
  - Module
- Relationen
  - *Is part of*
  - *Depends on*
  - *Is a*
  - *Uses …*

Konzeptioneller Überblick



Module Viewtype - UML



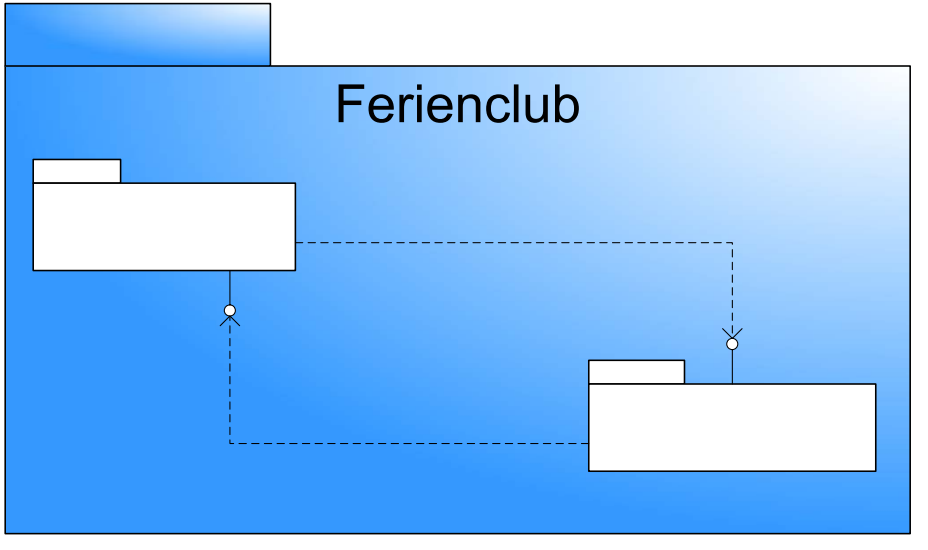
Key: UML

Servicebus Architektur im Ferienclub

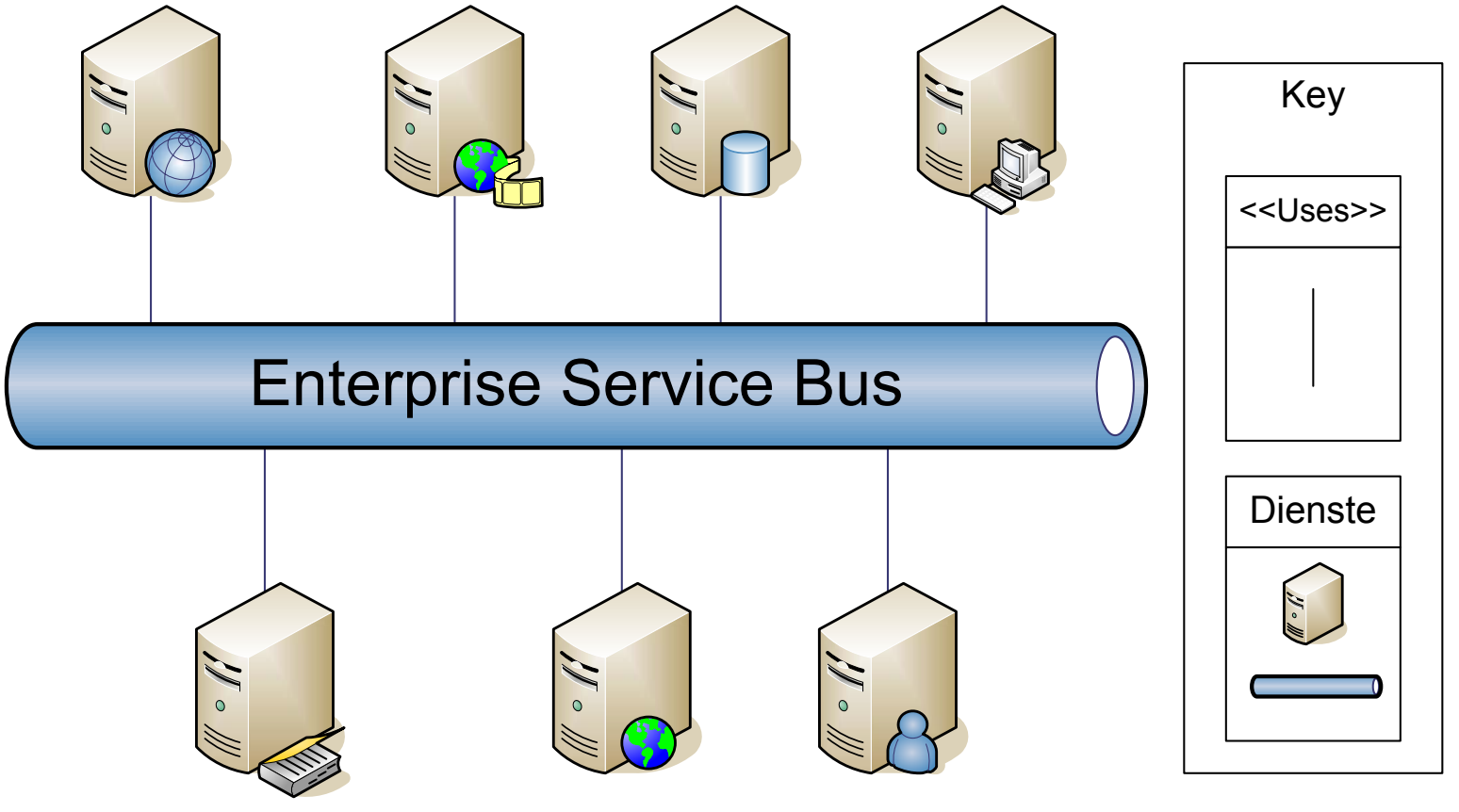
Uses-Style

Module Viewtype - ACME

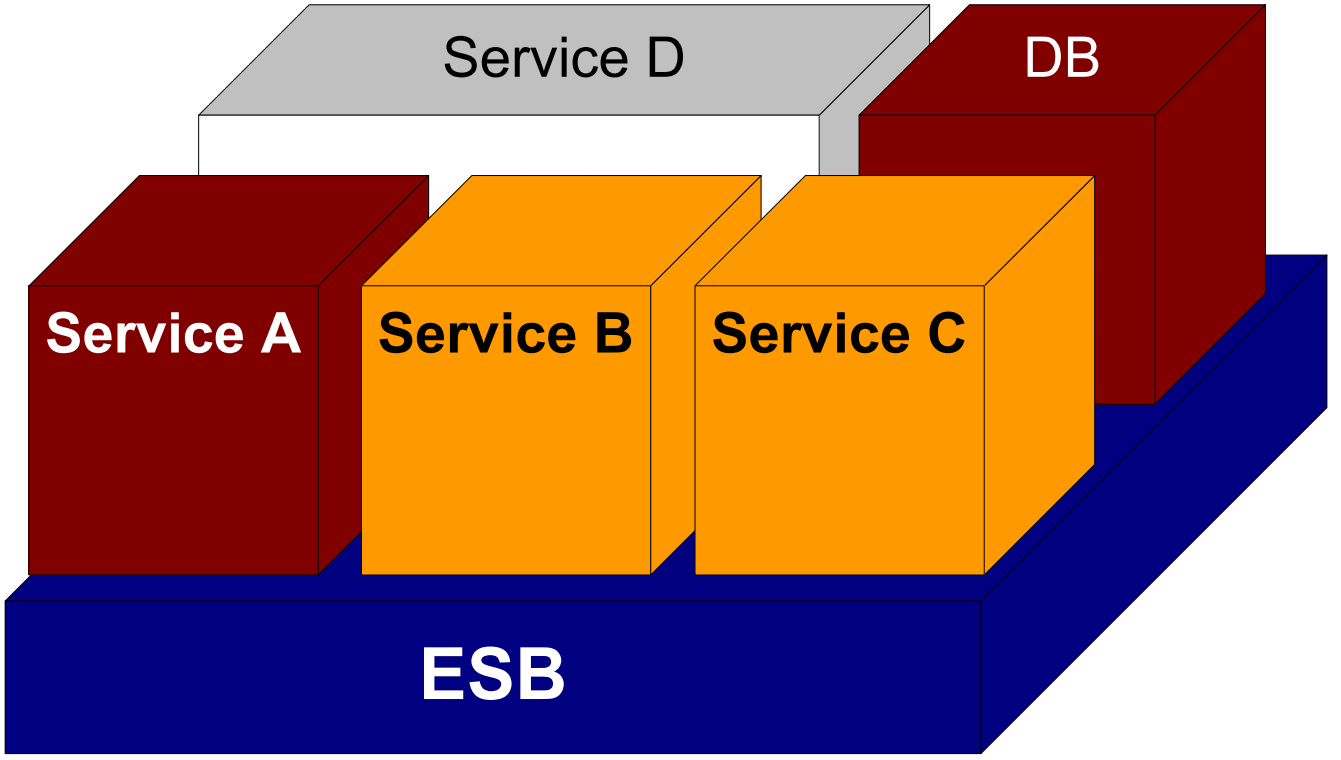
```
System Ferienclub = {
    Component enterpriseServiceBus = {
        Port outBound;
        Port inBound;
    };
    Component service = {
        Port outBound;
        Port inBound;
    };
    Connector upLink = {
        Role caller;
        Role callee;
    };
    Connector downLink = {
        Role caller;
        Role callee;
    };
    Attachment enterpriseServiceBus.outBound to downLink.caller;
    Attachment service.inBound to downLink.callee;
    Attachment enterpriseServiceBus.inBound to upLink.callee;
    Attachment service.outBound to upLink.caller;
};
```



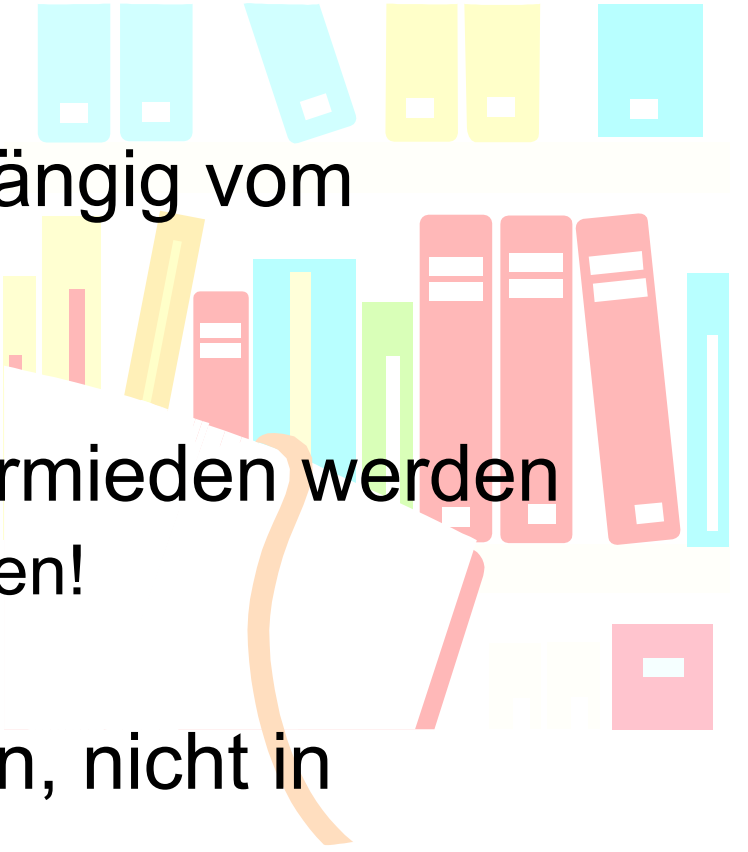
Module Viewtype - Informal



Module Viewtype - Informal



Module Viewtype



- Wahl der Notation abhängig vom Stakeholder
- „Overloading“ sollte vermieden werden
  - Wo möglich, abstrahieren!
- In Architekturen denken, nicht in Datenstrukturen!

Zusammenfassung

- Module sind die Grundbausteine
- Im *Module Viewtype* werden die Module und ihre Beziehungen untereinander dokumentiert

Component-and-Connector

- Dokumentation von
  - Elementen, die zur Laufzeit existieren
  - Flüssen von Informationen
  - Verhalten/Interaktion
- Elemente
  - Komponenten
  - Konnektoren
- Relationen
  - *attachment*
  - Verbinden Komponenten mit Konnektoren

C&C Komponenten

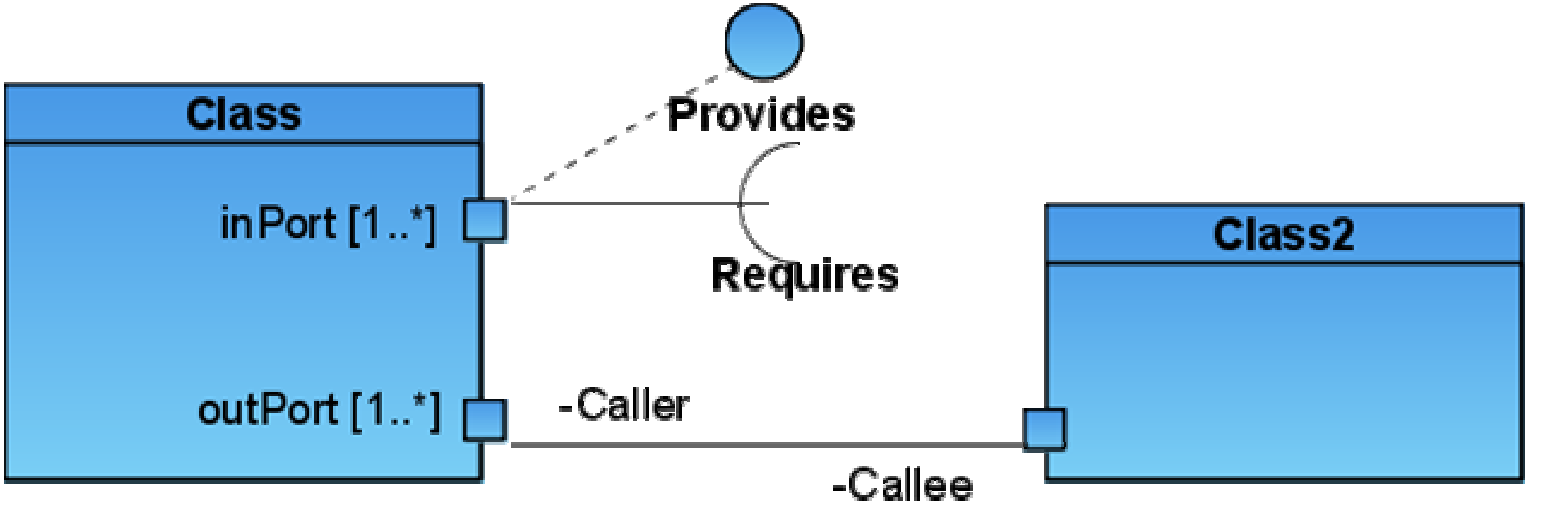
- Komponenten sind für Applikationslogik oder Speicherung zuständig
- Komponenten kommunizieren mit der Umgebung über ihre *Ports*

C&C Komponenten - Ports

- Interfaces der Komponenten
- Keine Interfaces im Sinne von UML
  - Interfaces können in UML nicht instantiiert werden
  - Trennung zwischen Interface und realisierender Klasse
- Gebraucht wird die semantische Trennung von Port und Komponente

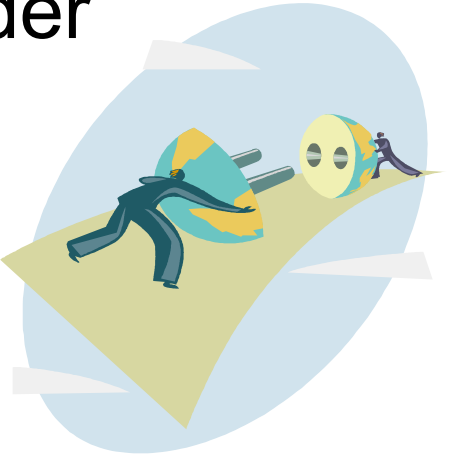
C&C Komponenten - Ports

- Diagrammtyp: UML 2.0 Composite Structure
  - Konnektoren
  - Ports
  - Provides / Requires

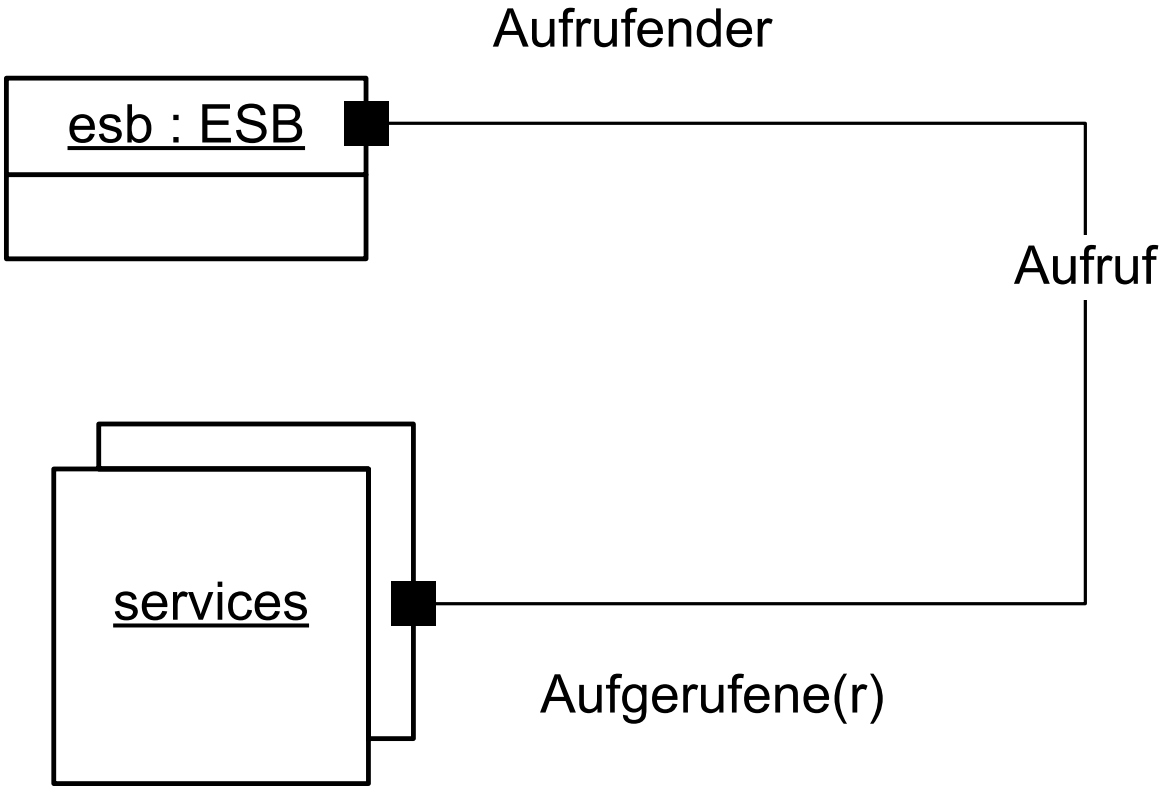


C&C - Konnektoren

- Interaktion, die zur Laufzeit stattfindet
- Steht für eine spezifische Form der Kommunikation
- Interface zu den Komponenten:
  - Rolle
- *Beispiele*
  - Transaktionen, RPC, Multicast-Messaging, …

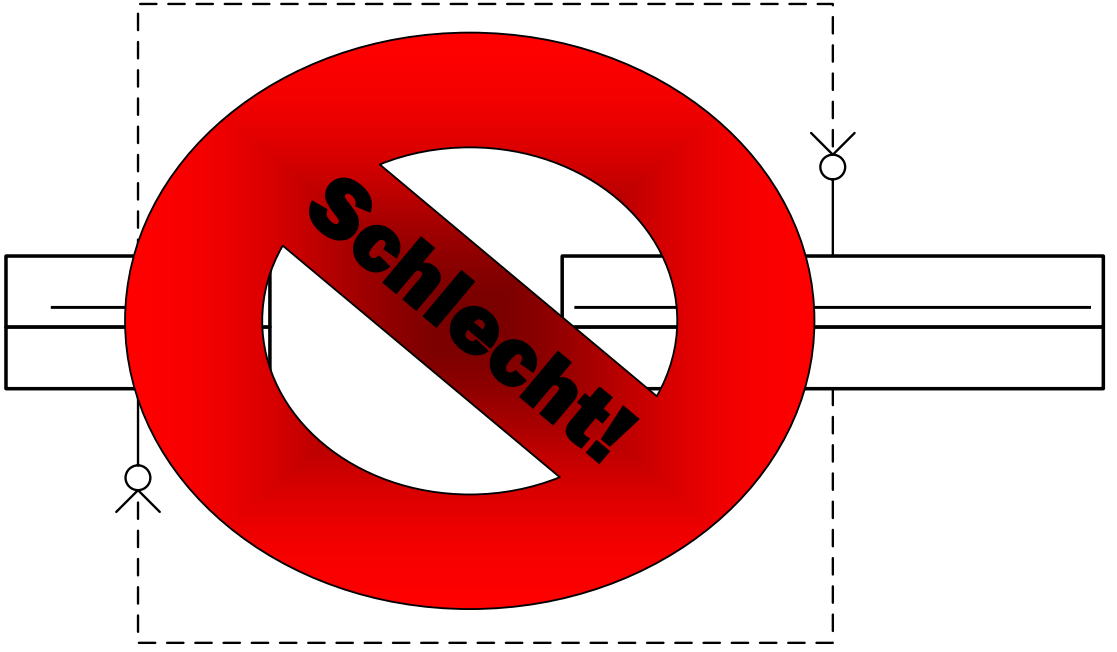


Rollen von Konnektoren



Key: UML

C&C – Naiv mit UML



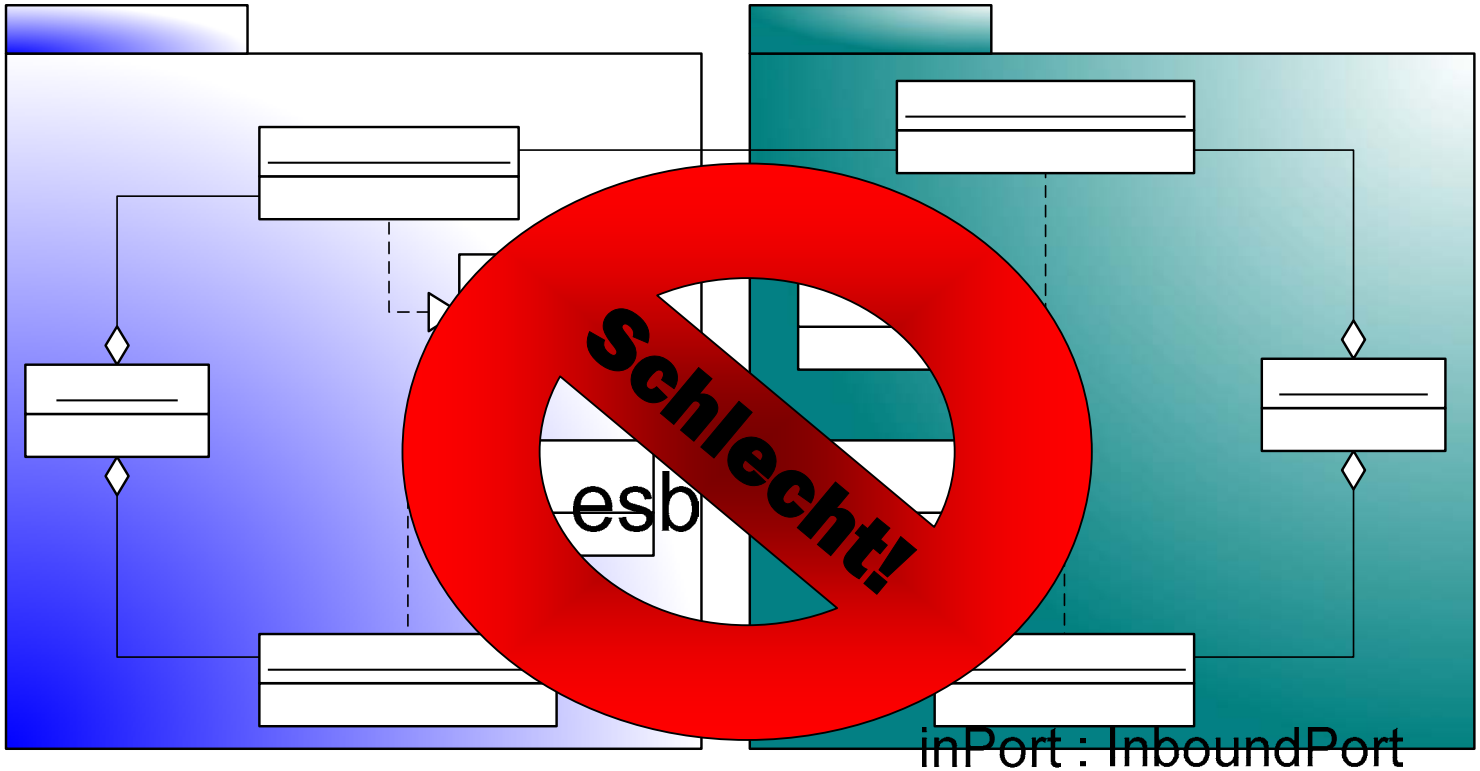
«uses»

Key: UML

Servicebus Architektur im Ferienclub

C&C-Style

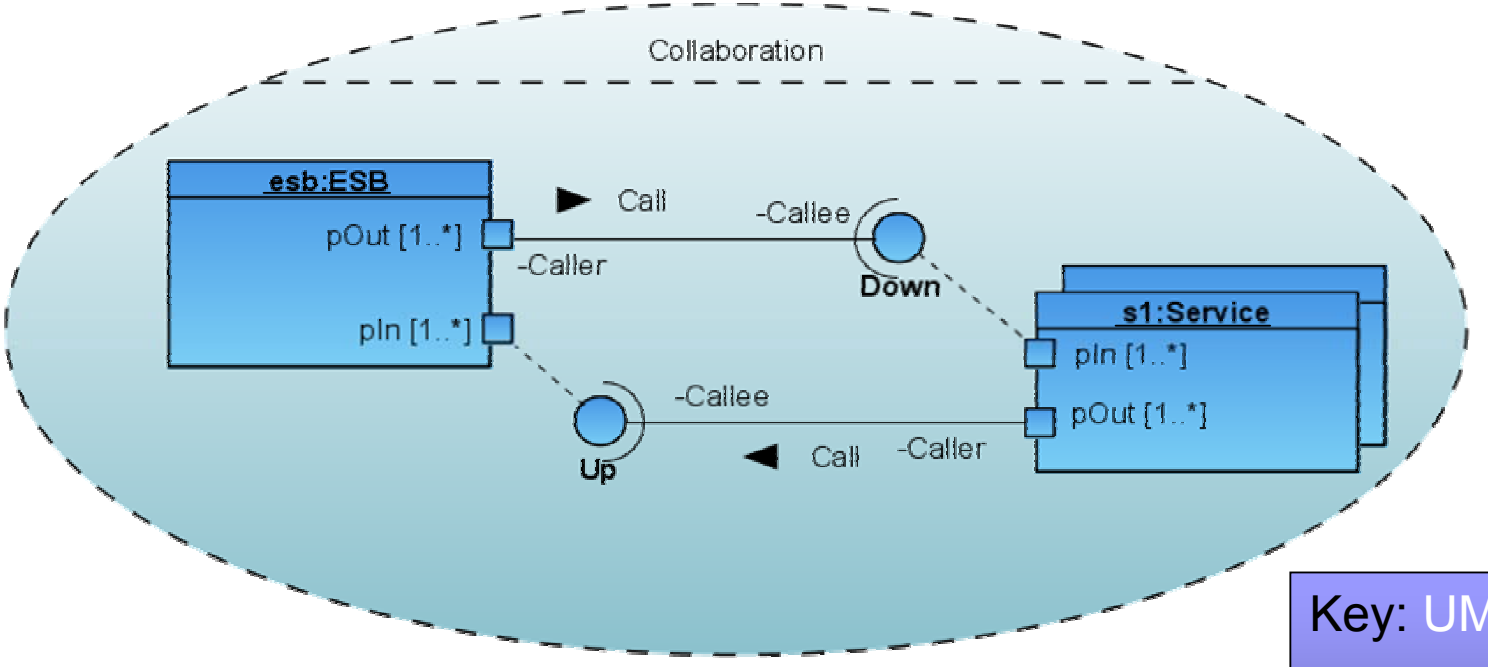
C&C – Overloading mit UML



Key: UML Servicebus Architektur im Ferienclub – C&C-Style

«Schnittstelle

C&C – UML 2.0



Key: UML2.0

Servicebus Architektur im Ferienclub

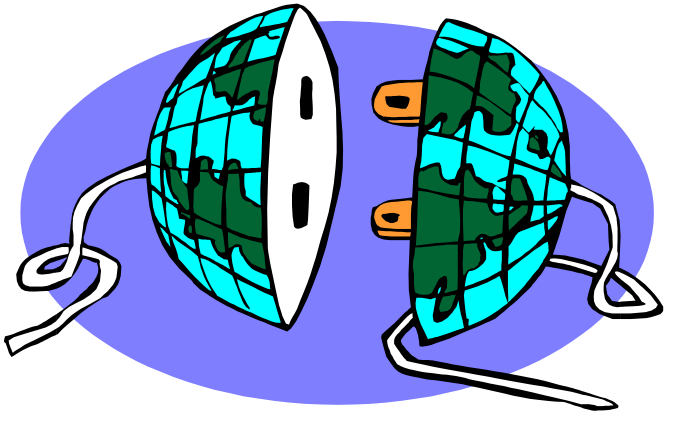
C&C-Style

Interfaces

- Treffpunkt zweier unabhängiger Entitäten, an dem sie miteinander interagieren oder kommunizieren [CBB+03]
- Jedes Element einer Architektur hat Interfaces

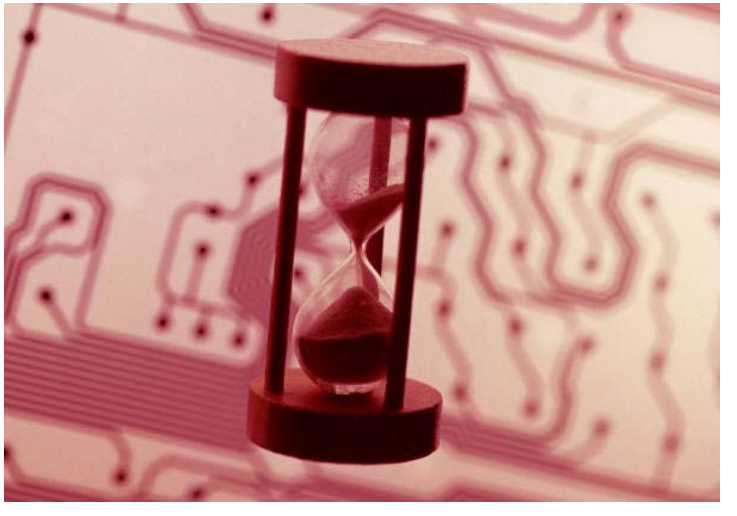
Interfaces

- Was muss kommuniziert / dokumentiert werden?
  - Syntax / Signatur
  - Semantik
  - Ausnahmen / Fehler
  - Pre-/Postconditions
  - Qualitätsattribute
- Wie läuft die Interaktion ab?
  - Nicht: Wie ist die Implementierung?



Zusammenfassung

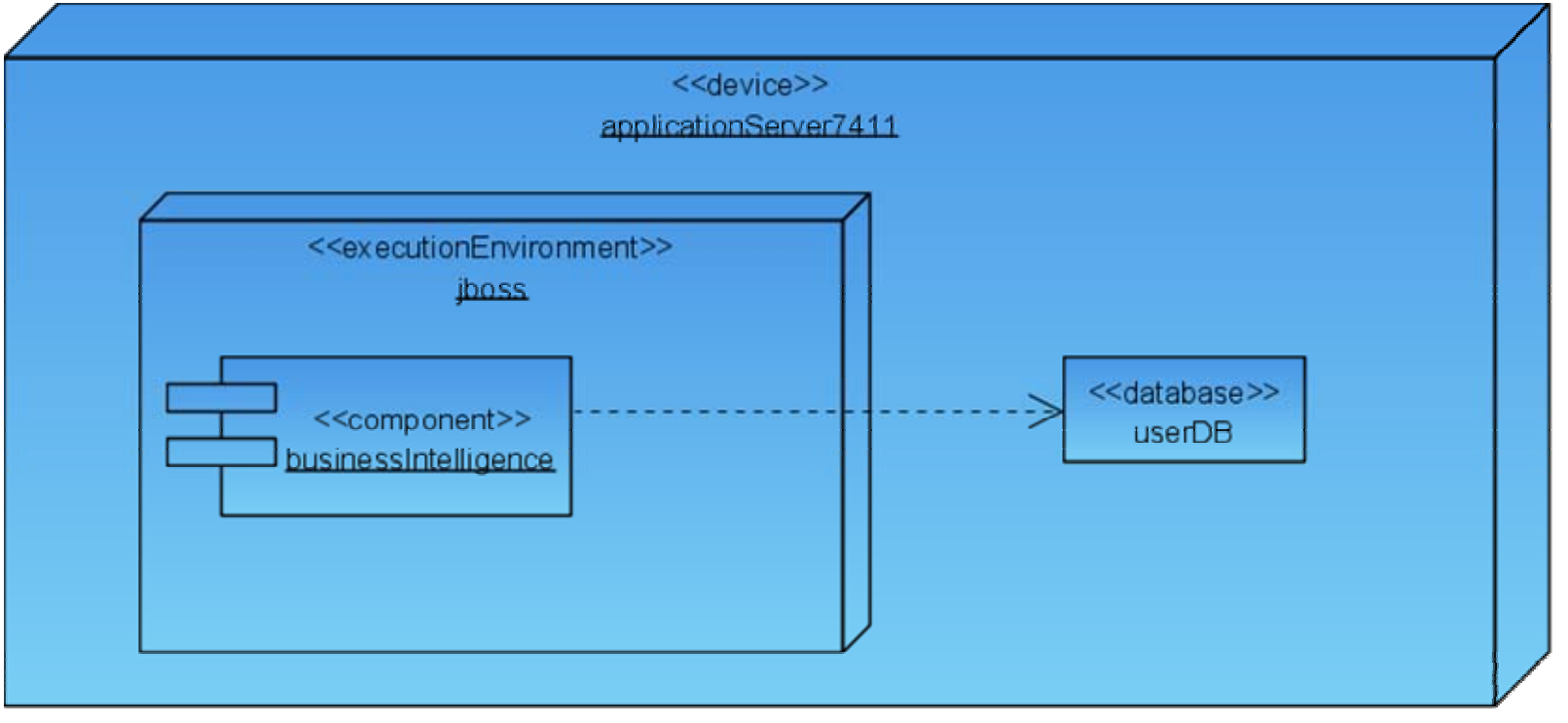
- Der C&C Viewtype dokumentiert das Laufzeitverhalten einer Architektur
  - Arbeitende Elemente
  - Kommunikation
  - Ressourcenzugriffe
- Wichtig, wenn es um Performance und Verfügbarkeit geht



Allocation Viewtype

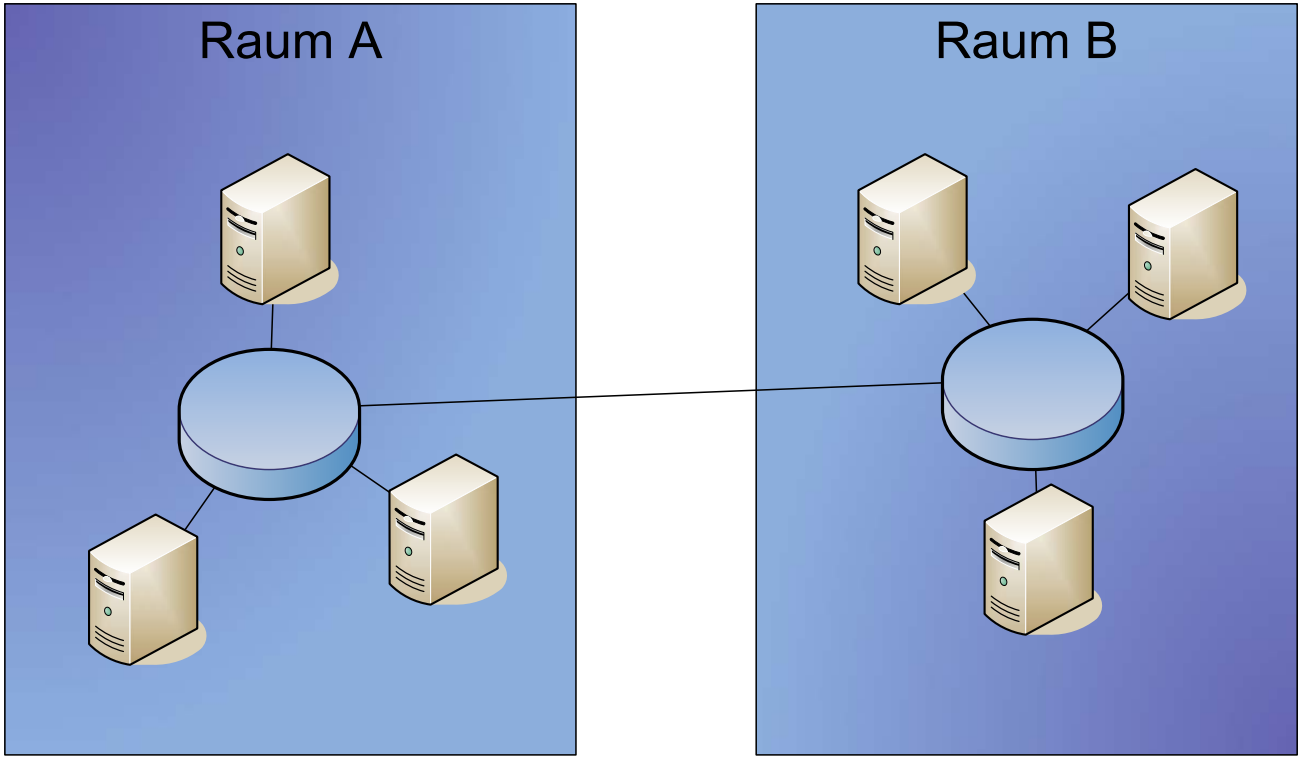
- Dokumentation
  - der Architektur und der Umgebung
  - der Nutzung von Hardware
  - der Zuständigkeiten
- Grafische Notation nicht immer notwendig
- Elemente
  - Software Elemente
  - Hardware
  - Dateisysteme
  - Personen/Gruppen
- Relationen
  - *Allocated-to*
  - *Responsible*

Allocation Viewtype



Key: UML – Deployment eines Webservices

Allocation Viewtype



Key: Informal

Zusammenfassung

- Allocation ► Mapping der Software Architektur auf ihre Umgebung und beteiligte Personen

- Zuständigkeiten
- Hardware
- Räumlichkeiten

Masterarbeit

Masterarbeit

- Evaluierung von
  - ADLs
  - Dokumentationsverfahren

  im Kontext *Verteilte Systeme*

- Tauglichkeit / Untauglichkeit für Aspekt x
  - Verbesserungsvorschläge

Fragen

Literatur

- **[BCK03]** Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Series in Software Engineering. Addison Wesley, 2. Edition, 2003.
- **[BKB02]** Leonard J. Bass, Mark Klein, and Felix Bachmann. Quality attribute design primitives and the attribute driven design method. In *PFE '01: Revised Papers from the 4th International Workshop on Software Product-Family Engineering*, pages 169-186, London, UK, 2002. ACM, Springer-Verlag.
- **[BKM01]** Len Bass, Mark Klein, and Gabriel Moreno. Applicability of general scenarios to the architecture tradeoff analysis method. Technical Report CMU/SEI-2001-TR-014, School of Computer Science - Carnegie Mellon University, 2001.

Literatur

- **[CBB+03]** Paul Clements, Felix Bachmann, Len Bass, et al. *Documenting Software Architectures*. Series in Software Engineering. Addison Wesley, 1. edition, 2003.
- **[Cle05]** Paul Clements. Comparing the sei's views and beyond approach for documenting software architectures with ansi-ieee 1471-2000. Technical Report CMU/SEI-2005-TN-017, School of Computer Science - Carnegie Mellon University, Juli 2005.
- **[CKK02]** Paul Clements, Rick Kazman und Mark Klein: *Evaluating Software Architectures.* Series in Software Engineering. Addison Wesley, 1. Auflage, 2002.
- **[GMW97]** David Garlan, Robert Monroe, and David Wile. Acme: an architecture description interchange language. In *CASCON '97: Proceedings of the 1997 conference of the Centre for Advanced Studies on Collaborative research*, page 7. ACM, IBM Press, 1997.

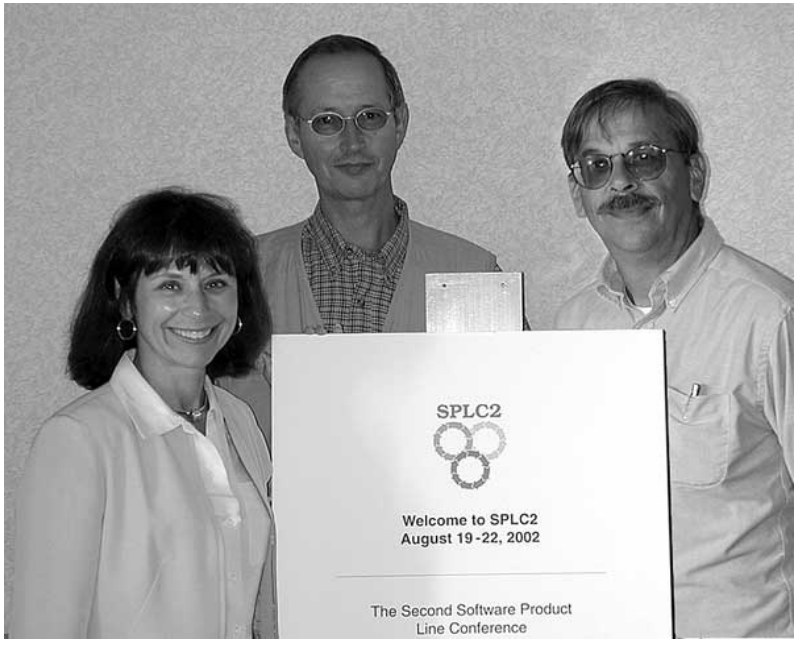
Literatur

- **[GS94]** David Garlan and Mary Shaw. An introduction to software architecture. Technical Report CMU-CS-94-166, Carnegie Mellon University, January 1994.
- **[SEI05]** SEI: Carnegie Mellon Software Engineering Institute, http://www.sei.cmu.edu/architecture/. *Software Architecture for Software-Intensive Systems*, Oktober 2005.
- **[Sha01]** Mary Shaw. The coming-of-age of software architecture research. In *ICSE '01: Proceedings of the 23rd International Conference on Software Engineering*, page 656, Washington, DC, USA, 2001. IEEE Computer Society.

Literatur

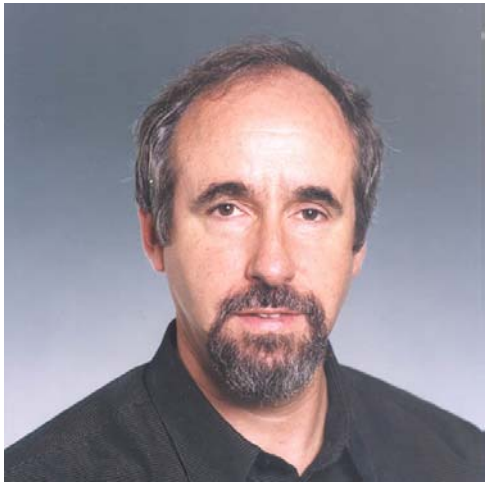
- **[SWLM02]** Mikael Svahnberg, Claes Wohlin, Lars Lundberg, and Michael Mattsson. A method for understanding quality attributes in software architecture structures. In *SEKE '02: Proceedings of the 14th international conference on Software engineering and knowledge engineering*, pages 819-826. ACM, ACM Press, 2002.
- **[IEEE 1471]** IEEE Std 1471 Recommended Practice for Architectural Description, 2000
- **[OMG_UML]** Definition von UML2.0 der OMG http://www.omg.org/technology/documents/formal/uml.htm
- **[CG01]** Cheng, Shang-Wen und David Garlan: Mapping Architectural Concepts to UML-RT. In: Proc. Parallel and Distributed Processing Techniques and Applications Conference, Las Vegas, NV, Seite 7. ACM, June 2001.

Köpfe



Linda Northrop, Henk Obbink, **Len Bass**

Rick Kazman



David Garlan